GreenCore: Green Computing for the Future of Defi

The GreenCore Team
xyz@supercompiler.xyz

Abstract

The blockchain paradigm has demonstrated its utility to unleash many new classes of decentralised applications, with DeFi as a prominent example. There have been a number of ongoing projects to address a key challenge in DeFi: the high transaction fees. We introduce a new solution, GreenCore, that alleviates this problem based on optimization theories of computer programs. GreenCore analyzes the deployed bytecodes of a smart contract and automatically optimizes its gas cost on every code path. We discuss its design, implementation and the opportunities it provides for the future. We further develop GreenSwap, by optimizing the popular Uniswap protocol, and find that it significantly reduces transaction fees on Ethereum by over 30% on average compared to Uniswap.

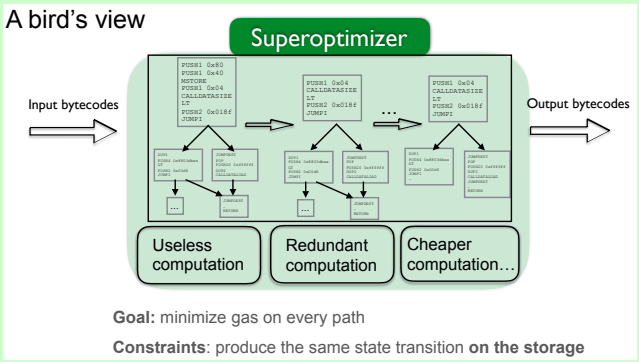

Figure 1. GreenCore.

1 Introduction

Blockchain technology has materialized the idea of building a world computer: a composable, open, permissionless state machine that runs trust-minimized code without borders. While we are still at an early stage of this revolution, DeFi (Decentralized Finance) has emerged as one of the most successful applications atop this new computer. Unlike traditional finance, DeFi allows anyone to borrow, invest, trade, issue and own digital assets without a centralised intermediary. This not only makes financing more efficient, but more importantly creates a transparent and more resilient financial system as compared to traditional banks.

In early 2021, the total market capitalization of DeFi has exceeded $100 billion. For many users, a pain in using DeFi is its high transaction fee, also known as "gas" fee. A simple exchange of two cryptocurrencies on Ethereum, regardless of the exchange amount, often costs more than $10, and can sometimes go up to over $1000. This problem has inspired a number of notable innovations to reduce transaction fees, such as Binance Smart Chain, Solana, as well as various Layer-2 solutions. However, none of them is perfect: they all rely on exploring different tradeoffs among three important elements—decentralization, security and scalability.

In this paper, we introduce a new solution in the design space—GreenCore—to alleviate this problem. GreenCore is based on optimization theories of computer programs. Observing that transaction fee is a product of two multiplication factors: gas price and gas cost of the transaction. While the former can fluctuate depending on the situation of network congestion, the latter is determined by the smart contracts, i.e., the code executed on the world computer during the transaction. GreenCore automatically optimizes the code towards reducing its gas cost, which consequently lowers the transaction fee by a multiplying factor. The more optimized the code is, the cheaper transaction fee we get.

Figure 1 depicts a bird's view of GreenCore, which takes a sequence of bytecodes as input, optimizes them with a Superoptimizer, and outputs another sequence of optimized bytecodes. Every smart contract is a sequence of bytecodes when deployed to the world computer. The output code from GreenCore has the same format as the input code and can be deployed with no changes to the world computer. Moreover, the output code is functionally equivalent to the input code: under all scenarios allowed by the world computer, the output code always produces the same state transition as the input code (note that the computer is a state machine).

GreenCore's cornerstone is an automated Superoptimizer driven by *Maximal Concolic Execution (MCE)*, a code verification technique developed by the programming language research community. MCE provides the ability to systematically explore every code path, encode path conditions with mathematical formulas, and check path invariants with SMT (Satisfiability Modulo Theories) solvers. The Superoptimizer can then identify optimization opportunities, such as useless computation, redundant computation, cheaper computation and so on, to minimize gas cost on every code path. Finally, the optimization is realized by iteratively transforming an intermediate graph representation of the input code while respecting the path invariants.

Powered by GreenCore, we have developed GreenSwap, a highly optimized DEX based on Uniswap—an automated market maker (AMM) on Ethereum. Uniswap is among the most popular DeFi protocol for swapping ERC20 tokens, and

GreenSwap: 1,199 bytes

0x60043611610009570056044361061031d57426024351161004d5762461bcd60e51b6080526020608452601260a4527111dc99595b94ddd85c0
e8811561412549151607216b60c45260646080fd5b6004356080630240bc6b60e21b81527339444e8ee494c6212054cfadf67abdbe97e702076060
82600484845afa50815160a0516000803560e01c635ed1a13f811461009f5763ae76be7681146101d057005b6100aa84843461036e565b8088111
56100ee5762461bcd60e51b875260206084526 01e60a4527f477265656e537761703a20494e5355464649434 9454e545f4f5554505554000060c4
52606487fd5b6370a0823160e01b87523060845273b93152b59e65a6de8d3464061bcc1d68f6749f98602060a460248a845afa508160a45110610
1445763a9059cbb60e01b88523360845260a4829052602088604481878 55af1005b5073c778417e063141139fce010982780140aa0cd5ab602088
60248a845afa5034885110156101825763 0d0e30db60e41b88528380600 48a47855af1505b63a9059cbb60e01b885260848790523460a45260208
860448187855af1505063022c0d9f60e01b8752806084525081 60a4523360c452602060e4528182608488858 95af16101ce578182fd5b005b6064
36146101dc578182fd5b6044356101ea84868361036e565b80891115610 22e5762461bcd60e51b8852602060845260 1e60a4527f477265656e537
761703a20494e5355464649434 9454e545f4f5554505554000060c452606488fd5b804710610281576323b872dd60e01b8852336084523060a452
60c4829052838060648a8273b93152b59e65a6de8d3464061bcc1d68f6749f985af1610271578384fd5b8384858684335af16101ce578384fd5b6
323b872dd60e01b8852336084528660a4528160c452838460648a8773b93152b59e65a6de8d3464061bcc1d68f6749f985af16102bc578384fd5b
63022c0d9f60e01b8852836084528060a4523060c4528360e452838460848a878b5af150632e1a7d4d60e01b8852608481905283806024 8a8273c
778417e063141139fce010982780140aa0cd5ab5af1508384858684335af16101ce578384fd5b6000543314156101ce57600435600035 60e01c63
a6f9dae181146103535763 51cff8d9811461035957632909 2d0e811461036257005b81600055005b6101ce8261040c565b61036b8261040c565b8
1ff5b6000816103b35762461bcd60e51b6080526020608452601d60a4527f477265656e537761703a20494e5355464649434 9454e545f494e5055
5400000060c452606460 80fd5b6103e5808302806103e886020181811 6103fc5762461bcd60e51b6080526020608452601460a4527364732d6d6
174682d6164642d6f766572666c6f7760601b60c4526046080fd5b9390950202919091049392505050565b60008081828347865af1506370a082
3160e01b6080818152306084527 3b93152b59e65a6de8d3464061bcc1d68f6749f986020826024818 45afa50815163a9059cbb60e01b8084528 66
08452816 0a4528586604486898 75af150938352306084527 3c778417e063141139fce010982780140aa0cd5ab936020846024818 85afa50835190
8452608487905260a4525083905080604483828 65af18556

Uniswap: 22,398 bytes

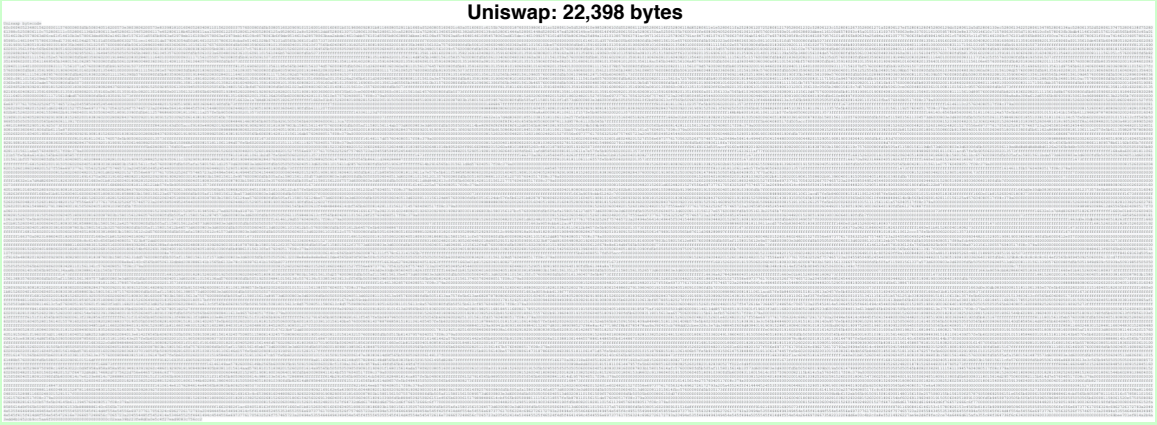

Figure 2. Deployed bytecodes of Greenswap and Uniswap.

its code has been executed over 50 million times by millions of users through swap transactions. A swap transaction on Uniswap takes about 150K gas cost and $20 gas fee on average. Compared to Uniswap, GreenSwap reduces the gas cost by as much as 55% (over 30% on average), saving as much as $10 fee per transaction. Figure 2 shows the bytecodes of GreenSwap and Uniswap (UniswapV2Router02). The size of GreenSwap is almost 18X smaller than Uniswap. This result indicates that there are abundant optimization opportunities in Uniswap identified by GreenCore. Note that the gas cost of a transaction is determined by the executed bytecodes on the code path during the transaction, not the size of the whole smart contract bytecodes. In our tests on Ethereum Ropsten, GreenSwap reduces the gas cost of Uniswap to only around 70K, more than 50% reduction on average. Once it is widely deployed and used by DeFi users, GreenSwap will save a significant amount of transaction fees.

We envision that GreenCore can serve as a foundation to make DeFi more efficient and ESG-friendly (Environmental, Social, Governance). GreenCore has also a large potential to be used for ensuring security of smart contracts. When used for auditing purposes, it can eliminate the gap of compiler bugs, which may introduce new security vulnerabilities in the compiled bytecodes unseen from the source code.

2 The GreenCore Technology

We use a simple flipper contract written in Solidity and compiled for the EVM to illustrate the GreenCore technology.

2.1 EVM

The Ethereum Virtual Machine (EVM) is a stack machine that executes all the bytecodes deployed on Ethereum. Most bytecodes compute data on the stack, or read/write data on a temporal memory or the persistent storage which is expensive. There are a few control bytecodes that determine what to execute next, such as JUMP/JUMPI/CALL/STOP/RETURN/REVERT. The first step of GreenCore is to transform the input sequence of bytecodes into an intermediate graph representation that respects the execution flow determined by the control bytecodes. Figure 3(a) shows the source code of the flipper contract. Figure 3(b) shows the bytecodes generated by the Solidity compiler with optimization turned on. Figure 3(c) shows a graph constructed from the bytecodes. Each node in the graph contains a subsequence of the input, starts with a tag (a jump target excepts the entry node) and ends with a control bytecode. Each edge in the graph represents the execution flow from a control bytecode to a jump target. Note that the graph may not be straightforward to construct, when a jump

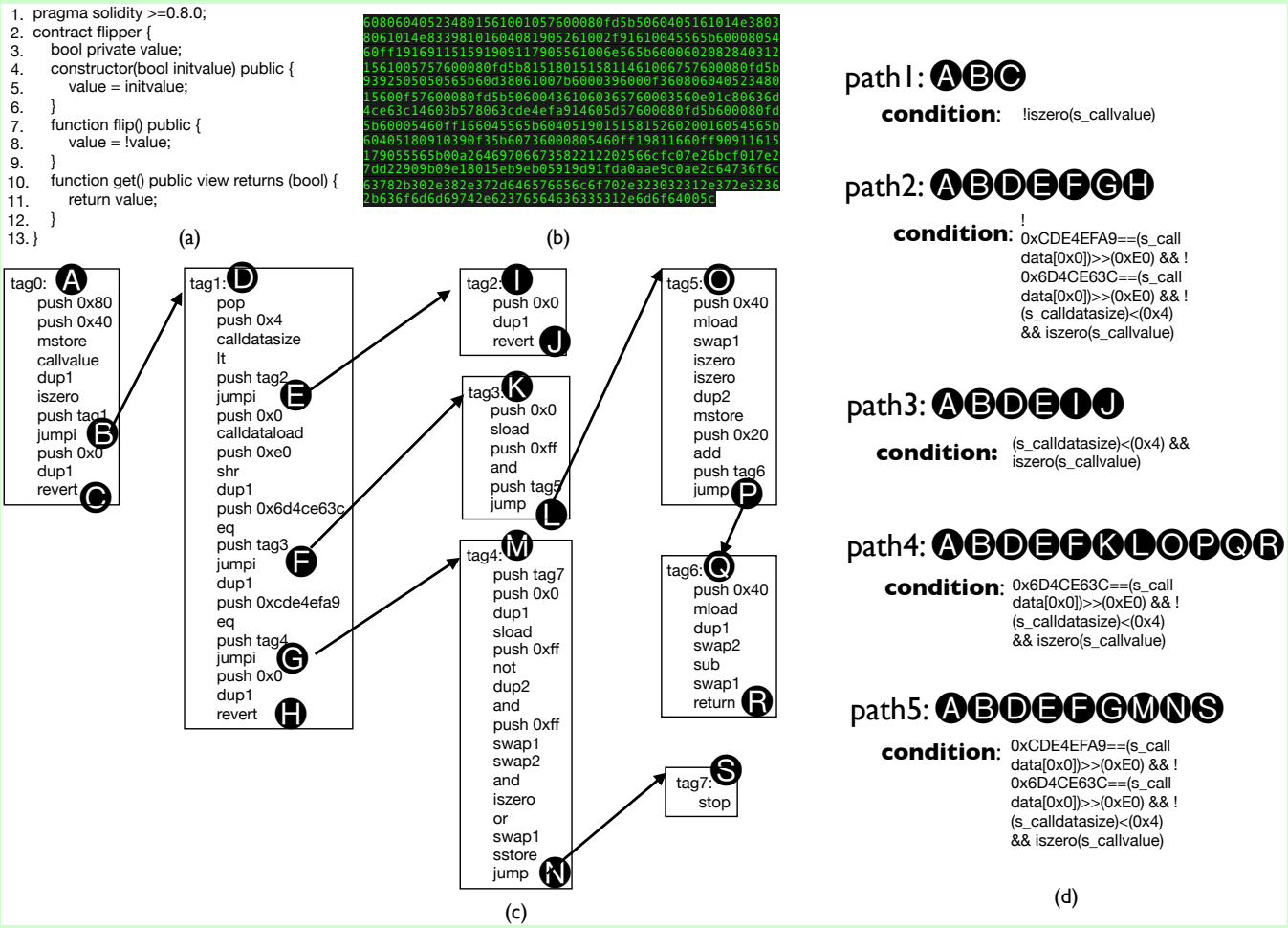

Figure 3. An illustration of GreenCore: (a) a smart contract written in Solidity, (b) bytecodes produced by Solidity compiler with optimization, (c) graph representation of the bytecodes, (d) code paths and the corresponding path conditions.

target depends on computed data. This problem is addressed by Maximal Concolic Execution.

2.2 Maximal Concolic Execution

Maximal Concolic Execution (MCE) is a technique that uses both concrete values and symbolic values to efficiently test and verify code. It executes the input bytecodes by means of interpretation, with the goal to explore all code paths. Every code path starts from the input entry and ends with a terminating bytecode STOP/RETURN/REVERT. Whenever data is unknown, such as input data from the contract call or data returned from an external call, it will introduce a symbolic value to denote the unknown data, and the interpretation will continue using the symbolic values. Upon a control bytecode, MCE will determine the possible jump targets based on the path condition, which may contain symbolic values. The process is repeated until no new code paths can be found. Figure 3(d) shows the five paths explored by MCE for our example. Each path is associated with a path condition denoting that under what condition the path is valid. For example, `path1` has condition `!iszero(s_callvalue)`, meaning that this path will be executed when the contract is called with value. In more complex cases, invalid paths will be pruned by solving the corresponding path conditions with SMT solvers, which are often highly efficient.

2.3 Superoptimizer

The Superoptimizer identifies optimization opportunities based on the explored paths, and performs the optimizations by transforming the graph representation. The optimization opportunities can be charaterized with respect to removing useless and redundant computation, and replacing expensive computations with equivalent but cheaper computation. Figure 4 shows a large number of optimizations identified in our example. For example, under `tag0` the first three bytecodes `push 0x80 push 0x40 mstore` can be removed because the data stored by `mstore` is never used. The `dup1` under `tag0` and `pop` under `tag1` are redundant and can be

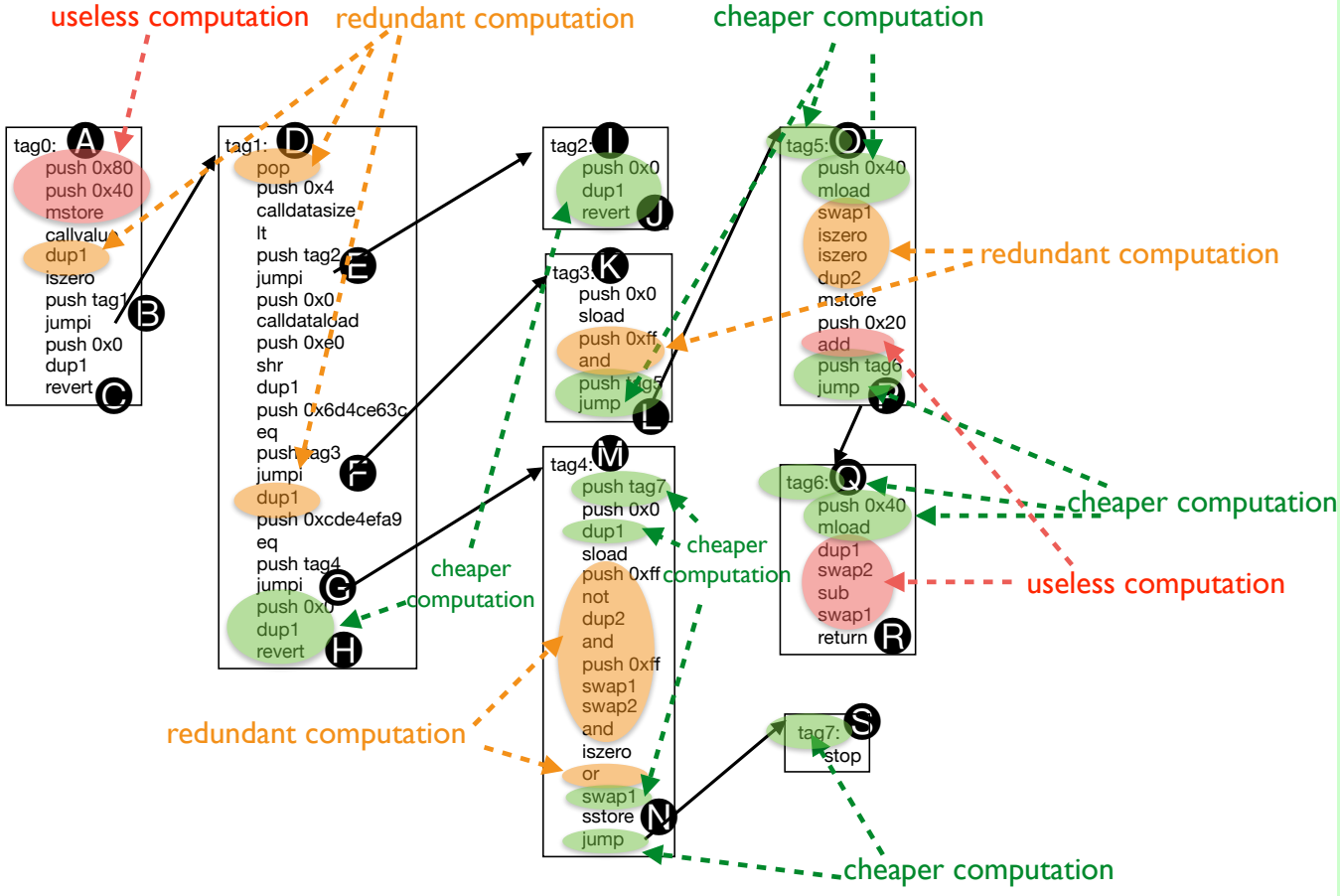

Figure 4. GreenCore Superoptimizer.

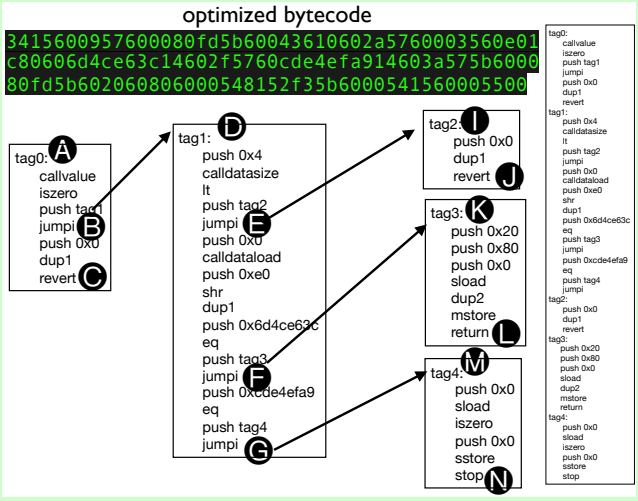

Figure 5. GreenCore optimized bytecodes.

both removed, because on `path1` the value by `dup1` is not used, and on all the remaining four paths the value is dropped by `pop`. The `jump` under `tag4` can be replaced by `stop` under `tag7`, which reduces the cost of a jump. The other cases are similar. After all the optimizations, GreenCore performs a topological sort of the graph (prioritizing hot code paths) and produces the output bytecodes by replacing tags with concrete addresses. Figure 5 shows the optimized bytecodes for our example. Compared to the input bytecodes in Figure 3(b), the size of optimized bytecodes is 4X smaller.

3 Conclusion and Future Directions

We have introduced GreenCore and illustrated the technical approach. We have also developed GreenSwap as one application of GreenCore. There are a number of future directions that are promising to further explore the potentials of GreenCore. In the immediate future, GreenCore can be applied more widely to other DeFi applications besides GreenSwap. GreenCore can be also used to audit and secure smart contracts by automatically verifying code paths at the level of bytecodes. In the longer term, GreenCore can be developed as a foundational tool to optimize and secure future decentralized applications and blockchains beyond Ethereum.